732A99 ComputerLab1 Block1

Namita Sharma

11/22/2019

Assignment 1. Spam classification with nearest neighbors

1.1 Import and divide data into training and test sets

The spambase.xlsx data in imported and divided into training and test sets in the proportion (50% / 50%) using R code detailed in appendix 1.1

1.2 Logistic model to predict spam using classification threshold 0.5

Table 1: Confusion Matrix Training Data

	0	1
0	804	127
1	93	346

Table 2: Confusion Matrix Test Data

	0	1
0	808	143
1	92	327

Misclassification Rates

	Training	Test
Misclassification Rate	0.1605839	0.1715328

The misclassification rate for the training data is slightly better than the misclassification rate for the test data.

1.3 Logistic model to predict spam using classification threshold 0.8

Table 3: Confusion Matrix Training Data

	0	1
0	921	10
1	333	106

Table 4: Confusion Matrix Test Data

	0	1
0	931	20
1	314	105

Misclassification Rates

	Training	Test
Misclassification Rate	0.250365	0.2437956

Although the classification 0.5 has better misclassification rates on both training and test data sets when compared to the classification 0.8, it should be noted that the latter has fewer false positives (non-spam classified as spam) than the former.

Depending on the loss function and the penalty for classifying a non-spam email as spam (false positive) or a spam as non-spam (false negative), either of the models could be useful. If the loss in marking a non-spam email as spam is considered to be much greater than failing to detect a spam email, the classification 0.8 proves to be a better model.

1.4 K-Nearest Neighbours to classify spam emails using K = 30

Misclassification Rates

	Training	Test
Misclassification Rate	0.1671533	0.3131387

Table 5: Confusion Matrix Training Data

	0	1
0	779	152
1	77	362

Table 6: Confusion Matrix Test Data

	0	1
0	702	249
1	180	239

The misclassification rates of the kknn classification with k = 30 is similar to that of the logisticmodel with threshold = 0.5 for both training and test datasets. The two models are comparable in terms of their performance.

On the other hand, the kknn classification with k = 30 has lower misclassifcation rates for both training and test datasets when compared to the logistic model with threshold = 0.8.

As with the previous comparison, even though logistic classification 0.8 has a higher overall misclassification rate than kknn classification k = 30, it should be noted that the former has fewer false positives (detecting a non-spam as spam) and if the loss in classifying a non-spam email as spam is considered to be much greater than failing to detect a spam email, the classification 0.8 proves to be a better model overall.

1.5 K-Nearest Neighbours to classify spam emails using K = 1

Table 7: Confusion Matrix Training Data

	0	1
0	931	0
1	0	439

Table 8: Confusion Matrix Test Data

	0	1
0	644	307
1	185	234

Misclassification Rates

	Training	Test
Misclassification Rate	0	0.3591241

By reducing k, The misclassification rate using the kknn classification with k = 1 is zero for training data but very high for the test data. The model is overfitted to the training data and is poorer compared to the logistic model with threshold = 0.5 or 0.8 because it has higher misclassification rates on the test dataset.

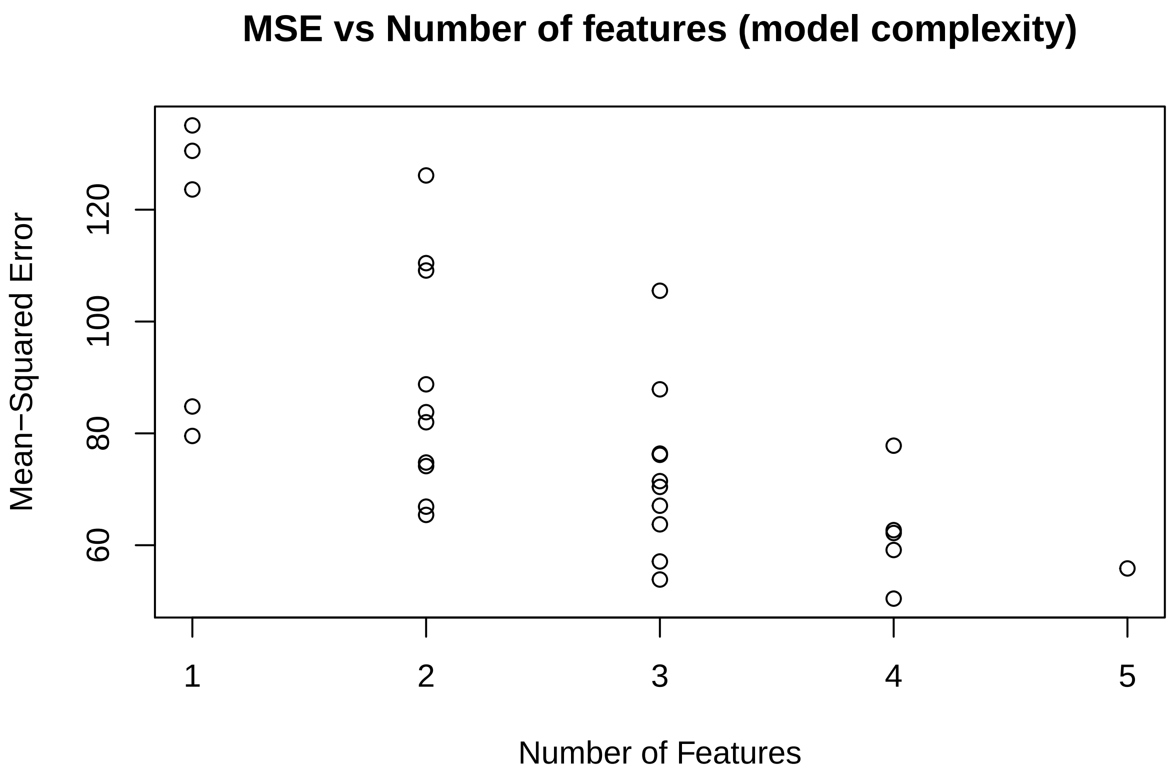
NOTE: The 0 misclassification rate in case of the training dataset can be attributed to the fact that the same dataset is used to train as well as test the model. Hence, every point in the dataset used for testing the model is closest to itself for k = 1 nearest neighbour classification.

Assignment 3. Feature selection by cross-validation in a linear model.

3.1 Implement an R function that performs feature selection (best subset selection) in linear regression by using k-fold

Cross validation in linear regression is implemented using R code detailed in appendix 3.1. The custom function calculates the CV scores for all subsets of features using k-fold cross validation in each linear model and determines the model with optimum subset of features having the least MSE.

3.2 Test implementation using swiss dataset



	CV	Features
Best Fit Model	50.44948	1 0 1 1 1

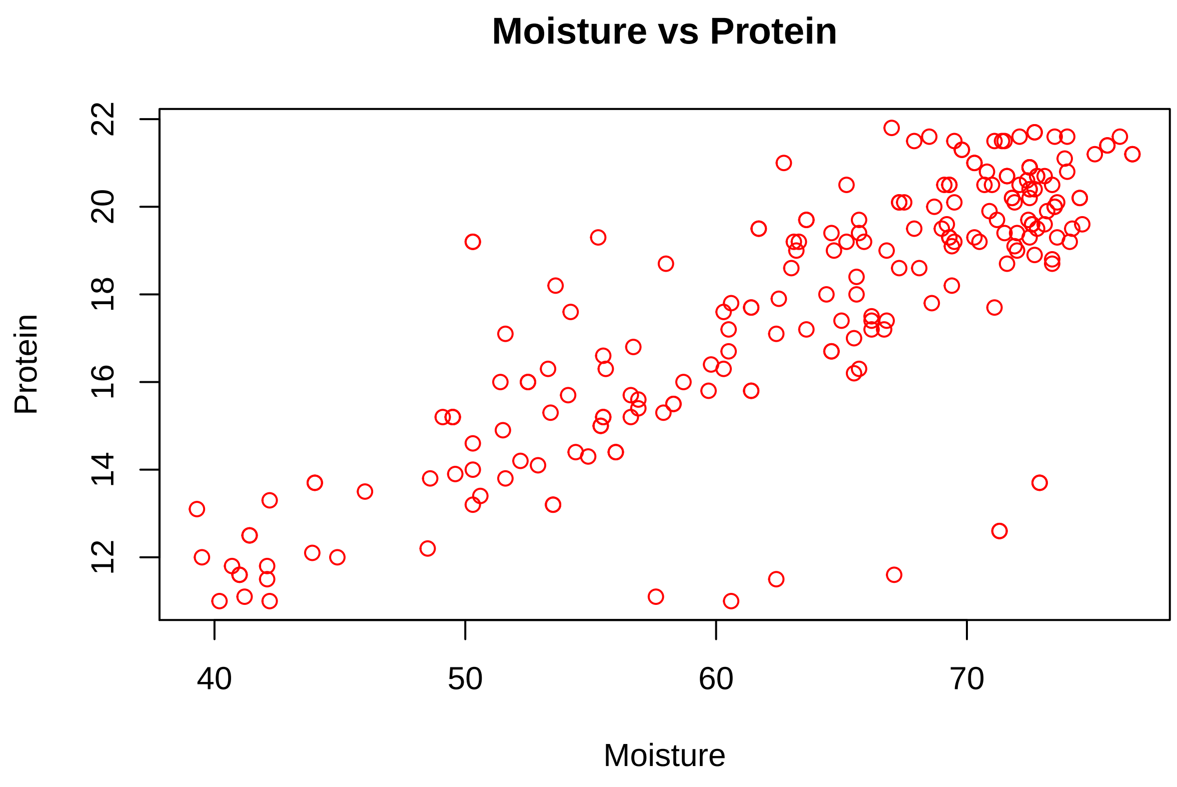
Selected Features = Agriculture, Education, Catholic, Infant.Mortality

The optimal model has a subset of 4 features - Agriculture, Education, Catholic and Infant.Mortality. It is reasonable that this subset of features has the largest impact on the target Fertility because this subset has the lowest MSE. This can be explained because the fifth feature Examination which is dropped in the best model is highly correlated with all the other features (predictors). High multicollinearity reduces the precision of the estimated coefficients (increases standard error) as it becomes harder to determine the effect of individual predictors on the target. Hence, eliminating this highly correlated feature results in better estimates of the target for changes in each of the remaining individual predictors and lowers the MSE of the model predictions.

It can also be observed from the graph that with an increase in the number of features/ predictors in the model (complexity), the MSE reduces.

Assignment 4. Linear regression and regularization

4.1 Import data and plot Moisture versus Protein

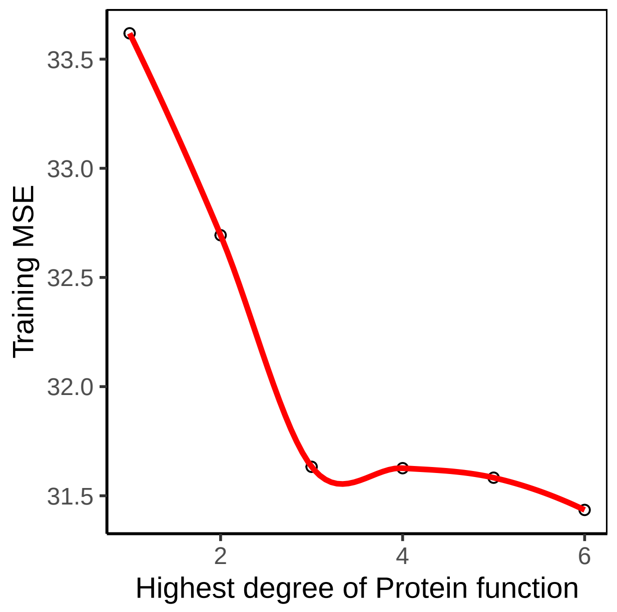


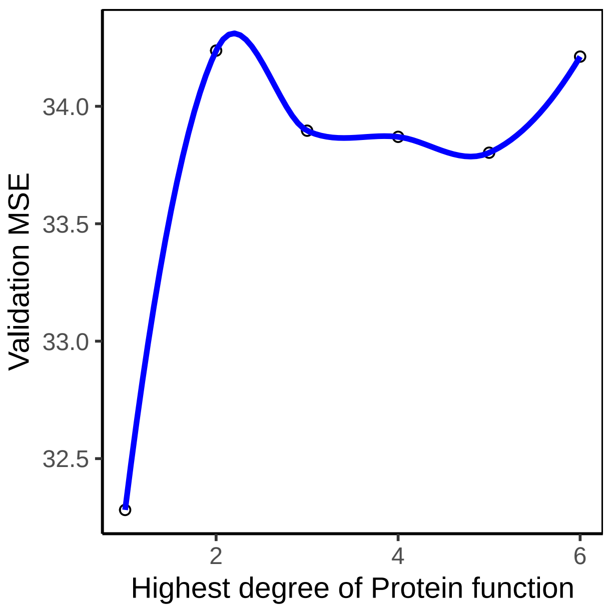
Since there seems to be a linear relationship between the Moisture and Protein data, it can be described well by a linear regression model.

4.2 Probabilistic model that describes Mi in which Moisture is a polynomial function of Protein including terms up to power i

The models Mi where $i = 1, M1 : Moisture = \beta 0 + \beta 1 Protein$ $i = 2, M2 : Moisture = \beta 0 + \beta 1 Protein + \beta 2 Protein^2$ and so on.. can be described by a linear regression model as the relationship between the target variable Moisture and the features $Protein$, $Protein^2$, etc. is linear. In order to maximize the likelihood of target y we need to minimize the MSE. Hence, MSE is a good criterion when fitting the models to training data.

4.3 Fit Models Mi, i = 1,2,3,4,5,6 for training and test datasets





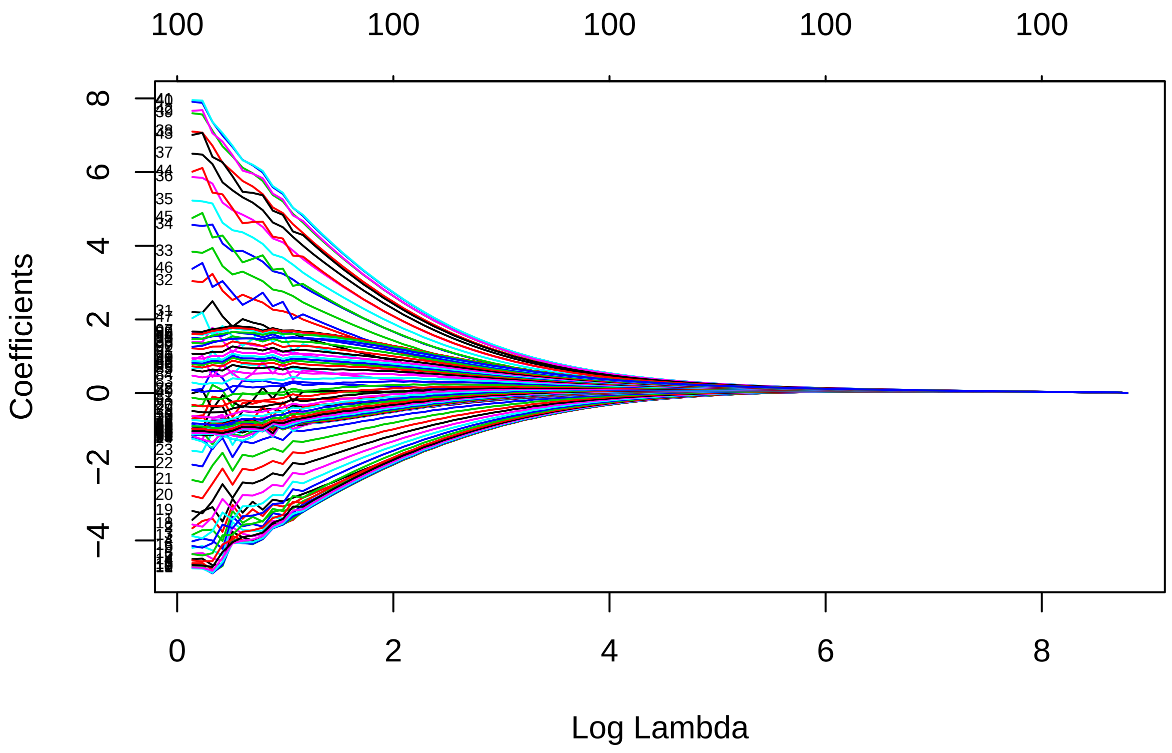
From the MSE plots, it is seen that for the training data, the MSE of the model decreases as new features are added to the model. But when the model is validated with the test data, the MSE of the model decreases only upto a certain increase in the number of features, after which adding new features limits the model performance on new test data. In this case models with features > 3 may fit the training data better, but do not generalize well to new test data. It can be said that Models with 1-3 features predict well on both training and test data.

4.4 Variable selection of a linear model Fat ~ Channel1-Channel100 using stepAIC

```
## Channel1 + Channel2 + Channel4 + Channel5 + Channel7 + Channel8 +
##     Channel11 + Channel12 + Channel13 + Channel14 + Channel15 +
##     Channel17 + Channel19 + Channel20 + Channel22 + Channel24 +
##     Channel25 + Channel26 + Channel28 + Channel29 + Channel30 +
##     Channel32 + Channel34 + Channel36 + Channel37 + Channel39 +
##     Channel40 + Channel41 + Channel42 + Channel45 + Channel46 +
##     Channel47 + Channel48 + Channel50 + Channel51 + Channel52 +
##     Channel54 + Channel55 + Channel56 + Channel59 + Channel60 +
##     Channel61 + Channel63 + Channel64 + Channel65 + Channel67 +
##     Channel68 + Channel69 + Channel71 + Channel73 + Channel74 +
##     Channel78 + Channel79 + Channel80 + Channel81 + Channel84 +
##     Channel85 + Channel87 + Channel88 + Channel92 + Channel94 +
##     Channel98 + Channel99
```

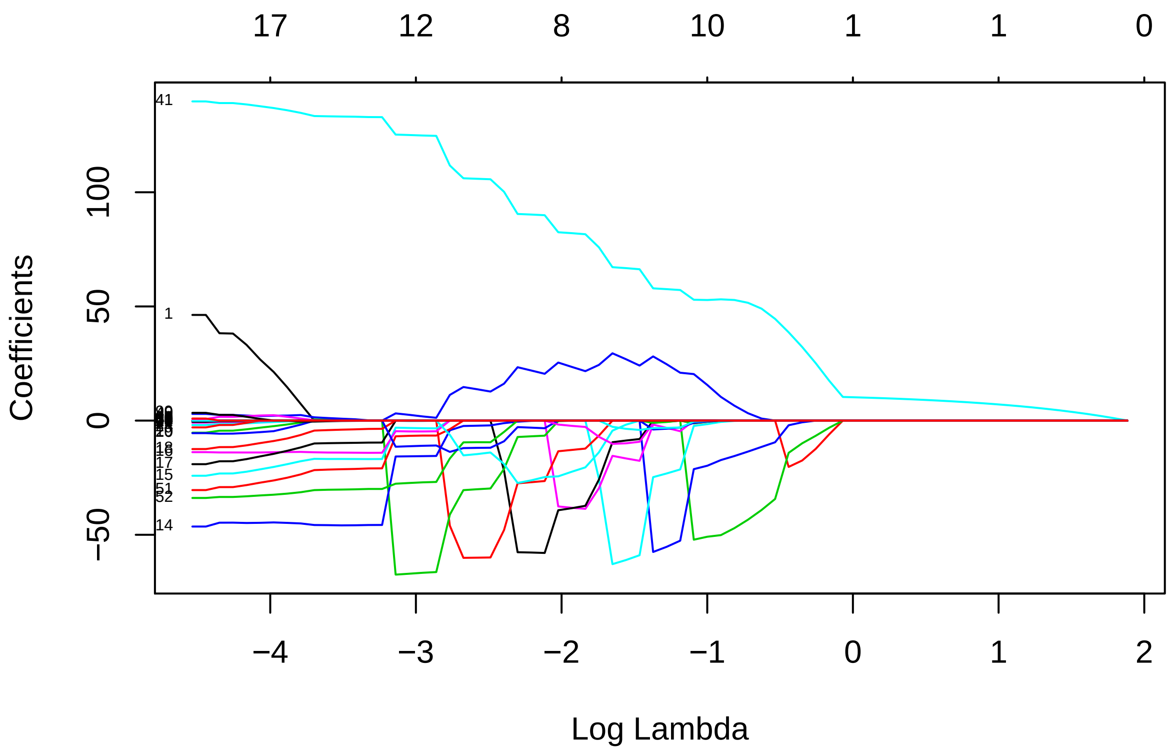
Using step AIC forward-backward selection, 63 out of the 100 predictors were selected in the model having the least AIC value (measure of information loss due to simplification). This means that by dropping the remaining 37 features out of the total 100 predictors, a much simpler model can be obtained without impacting much on the performance.

4.5 Fit a Ridge regression model Fat ~ Channel1-Channel100 for various penalty factor lambda



It is seen from the graph that as the penalty factor lambda is increased, the regression coefficients approach zero, i.e. the model becomes more and more simple with fewer significant features. Choosing an optimum value of lamdba would make some features less significant than the others by reducing the weights of their coefficients.

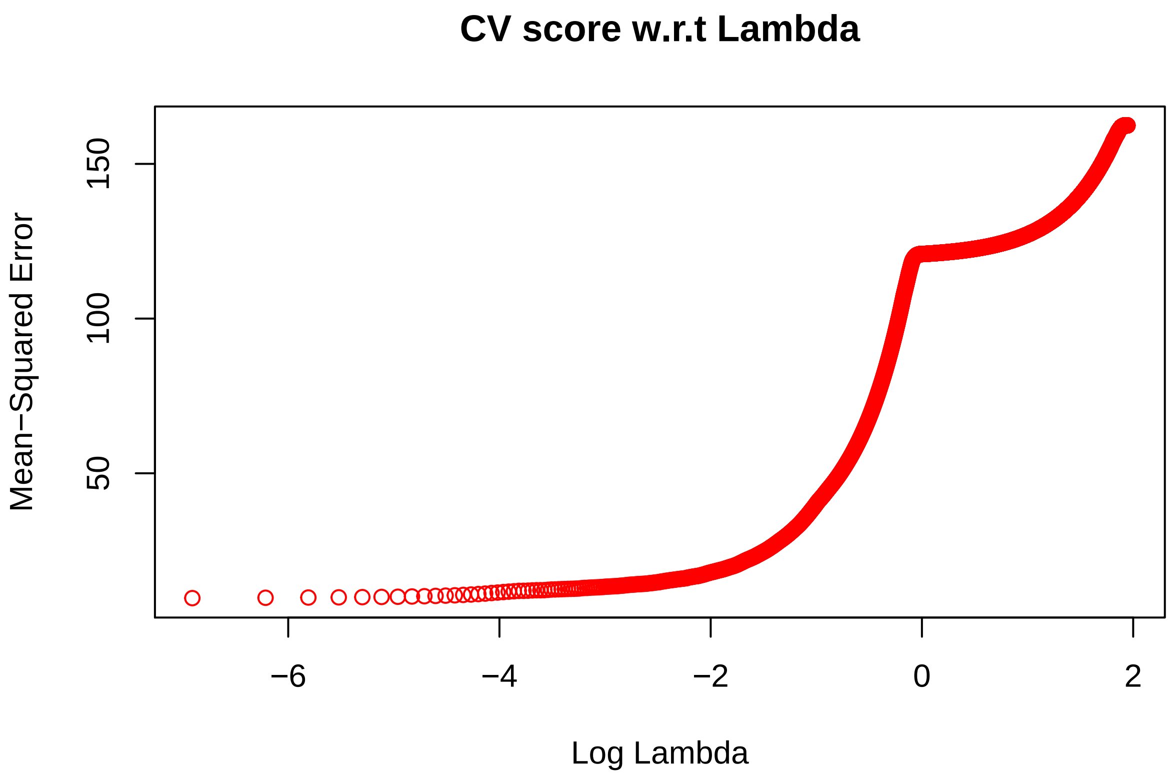
4.6 Fit a Lasso regression model Fat ~ Channel1-Channel100 for various penalty factor lambda



In case of lasso regression, coefficients shrink to zero as lambda value increases. It is similar to ridge regression but lasso eliminates some of the features by setting their regression coefficients to zero, i.e the model becomes more and more simple with fewer selected features.

Choosing an optimum value of lambda and thus selecting an optimal subset of features can prevent complex models which generally overfit to the training data, as seen in case of simple linear regression in exercise 4.3. When lamda = 0, the predicted coefficients are same as that of linear regression and as lambda -> Inf (lambda > 1 in this case), they shrink to zero. The optimum value of lambda can be chosen by performing cross validation lasso regression.

4.7 Cross-Validation to find the optimal LASSO model



	Lambda	Measure	SE	Nonzero
Min	0.000	9.488881	1.131688	100
1se	0.012	10.552218	1.013034	17

It can be seen from the graph that as the penalty factor λ increases, the MSE also increases. Combining the results from exercise 4.6, it can be said that as the lambda increases, more and more features are dropped from the model (coefficients shrink to zero) making the model simpler which results in a higher mean squared error.

Hence the lowest MSE recorded is 9.49 for $\lambda = 0$ (simple linear regression) where all of the 100 features are selected in the model. But while $\lambda.min$ gives the least MSE, it is clearly not the most optimum. A good model would be one which is not overly complex, but at the same time has low MSE with an optimal selection of features. It can be observed that $\lambda.1se = 0.012$ gives the largest subset of features such that the MSE is within 1 standard error of the minimum MSE. In other words, for a small increase in MSE value from the minimum MSE, the model is optimized by selecting only 17 features out of 100.

4.8 Compare results from 4.4 and 4.7

	Measure	Features
step AIC	0.8598985	63
CV Lasso	10.5522180	17

In stepAIC feature selection, optimum model is a subset of 63 features which has an MSE of 0.86. Whereas in CV lasso regression, the most optimum model has only 17 features.

While the MSE of stepAIC model is lower, it is based on the training data of the model, it may be much higher when tested on a new dataset. Whereas MSE of the CV lasso model is the validation MSE and works well with new data. It can be concluded that CV lasso is more optimum model as it is less complex with fewer parameters and has low MSE on test data.

Appendix

```r
##############################################################
# Assignment 1 - Spam classification with nearest neighbors
##############################################################

# 1.1 Import spam data from excel file
spambase <- xlsx::read.xlsx(
              file      = "C:/Users/namit/Downloads/Machine Learning/Lab1 Block1/spambase.xlsx",
              sheetName = "spambase_data",
              header    = TRUE)

# No of observations in the dataset
n <- dim(spambase)[1]

# Divide dataset into training and test data
set.seed(12345)
index <- sample(1:n, floor(n / 2))
train <- spambase[index, ]
test  <- spambase[-index, ]

# Create a logistic regression model using the training dataset
logistic.reg <- glm(formula = train$Spam ~ .,
                    data    = train,
                    family  = binomial(link = "logit"))

# Use the logistic model to predict spam in the training dataset
result_train <- predict(object  = logistic.reg,
                        newdata = train,
                        type    = "response")

# Use the logistic model to predict spam in the test dataset
result_test  <- predict(object  = logistic.reg,
                        newdata = test,
                        type    = "response")

#---------------------------------------------------------------------------
# 1.2 using classification threshold 0.5, compare the confusion matrices and
# misclassification rates for training and test datasets

train_pred1           <- ifelse(result_train > 0.5, 1, 0)
train_conf_mat1       <- table(actual = train$Spam, predicted = train_pred1)
train_misclass_rate1 <- 1 - (sum(diag(train_conf_mat1)) / sum(train_conf_mat1))

test_pred1            <- ifelse(result_test > 0.5, 1, 0)
test_conf_mat1        <- table(actual = test$Spam, predicted  = test_pred1)
```

```
test_misclass_rate1  <- 1 - (sum(diag(test_conf_mat1)) / sum(test_conf_mat1)))

#---------------------------------------------------------------------------
# 1.3 using classification threshold 0.8, compare the confusion matrices and
# misclassification rates for training and test datasets

train_pred2          <- ifelse(result_train > 0.8, 1, 0)
train_conf_mat2      <- table(actual = train$Spam, predicted = train_pred2)
train_misclass_rate2 <- 1 - (sum(diag(train_conf_mat2)) / sum(train_conf_mat2))

test_pred2           <- ifelse(result_test > 0.8, 1, 0)
test_conf_mat2       <- table(actual = test$Spam, predicted  = test_pred2)
test_misclass_rate2  <- 1 - (sum(diag(test_conf_mat2)) / sum(test_conf_mat2))

#---------------------------------------------------------------------------
# 1.4 Use kknn to classify spam emails in training and test datasets
# using k = 30 nearest neighbours

# Classify Spam in training dataset
kknn_train1 <- kknn::kknn(formula = Spam ~ .,
                          train   = train,
                          test    = train,
                          k       = 30)

# Classify Spam in test dataset
kknn_test1  <- kknn::kknn(formula = Spam ~ .,
                          train   = train,
                          test    = test,
                          k       = 30)

# Because there are only two classes, Spam = 1 and No Spam = 0,
# we classify the fitted values as 0 or 1 based on threshold
# probability 0.5
train_pred_1         <- ifelse(kknn_train1$fitted.values > 0.5, 1, 0)
train_conf_mat1      <- table(actual = train$Spam, predicted = train_pred_1)
train_misclass_rate1 <- 1 - (sum(diag(train_conf_mat1)) / sum(train_conf_mat1))

test_pred_1          <- ifelse(kknn_test1$fitted.values > 0.5, 1, 0)
test_conf_mat1       <- table(actual = test$Spam, predicted = test_pred_1)
test_misclass_rate1  <- 1 - (sum(diag(test_conf_mat1)) / sum(test_conf_mat1))

#---------------------------------------------------------------------------
# 1.5 Use kknn to classify spam emails in training and test datasets
# using k = 1 nearest neighbours

# Classify Spam in training dataset
kknn_train2 <- kknn::kknn(formula = Spam ~ .,
                          train   = train,
                          test    = train,
                          k       = 1)

# Classify Spam in test dataset
kknn_test2  <- kknn::kknn(formula = Spam ~ .,
```

```r
                           train   = train,
                           test    = test,
                           k       = 1)

# Because there are only two classes, Spam = 1 and No Spam = 0,
# we classify the fitted values as 0 or 1 based on threshold
# probability 0.5
train_pred_2          <- ifelse(kknn_train2$fitted.values > 0.5, 1, 0)
train_conf_mat2       <- table(actual = train$Spam, predicted = train_pred_2)
train_misclass_rate2 <- 1 - (sum(diag(train_conf_mat2)) / sum(train_conf_mat2))

test_pred_2           <- ifelse(kknn_test2$fitted.values > 0.5, 1, 0)
test_conf_mat2        <- table(actual = test$Spam, predicted = test_pred_2)
test_misclass_rate2   <- 1 - (sum(diag(test_conf_mat2)) / sum(test_conf_mat2))

###########################################################################
# Assignment 3 - Feature selection by cross-validation in a linear model
###########################################################################

# 3.1 Implement an R function that performs feature selection (best
# subset selection) in linear regression by using k-fold cross-validation

# Linear regression
mylin = function(X, Y, Xpred) {
  X1     <- cbind(1, X)
  Xpred1 <- cbind(1, Xpred)
  beta   <- solve(t(X1) %*% X1) %*% (t(X1) %*% Y)
  Res    <- Xpred1 %*% beta
  return(Res)
}

# Cross validation
myCV = function(X ,Y, Nfolds) {
  n <-length(Y)
  p <- ncol(X)

  set.seed(12345)
  ind      <- sample(n, n)
  X1       <- X[ind, ]
  Y1       <- Y[ind]
  sF       <- floor(n / Nfolds)
  MSE      <- numeric(2 ^ p - 1)
  Nfeat    <- numeric(2 ^ p - 1)
  Features <- list()
  curr     <- 0

  # We assume 5 features.
  for (f1 in 0:1)
    for (f2 in 0:1)
      for(f3 in 0:1)
        for(f4 in 0:1)
          for(f5 in 0:1) {
            model <- c(f1, f2, f3, f4, f5)
            if (sum(model) == 0) next()
            SSE <- 0

            for (k in 1:Nfolds) {
              # Compute which indices should belong to current fold
              index   <- (((k - 1) * sF) + 1):(k * sF)

              # Implement cross-validation for model with features in "model"
              # and iteration k.
              test_x  <- X1[index, which(model == 1)]
              train_x <- X1[-index, which(model == 1)]
              Yp      <- Y1[index]
              train_y <- Y1[-index]

              # Get the predicted values for fold 'k', Ypred, and the original
              # values for fold 'k', Yp.
              Ypred <- mylin(train_x, train_y, test_x)
              SSE   <- SSE + sum((Ypred - Yp) ^ 2)
            }
            curr             <- curr + 1
            MSE[curr]        <- SSE / n
            Nfeat[curr]      <- sum(model)
            Features[[curr]] <- model
          }

  # Plot MSE against number of features
  plot(Nfeat, MSE)

  # Best fit model
  i <- which.min(MSE)
  return(list(CV = MSE[i], Features = Features[[i]]))
}

#------------------------------------------------------------------------------
# 3.2 Test implementation using swiss dataset
myCV(as.matrix(swiss[, 2:6]), swiss[[1]], 5)

##########################################################
## Assignment 4 - Linear Regression and Regularization
##########################################################

# 4.1 Import tecator data from excel file
tecator <- xlsx::read.xlsx(
  file      = "C:/Users/namit/Downloads/Machine Learning/Lab1 Block1/tecator.xlsx",
  sheetName = "data",
  header    = TRUE)

# Plot Moisture vs Protein
plot(tecator$Moisture, tecator$Protein, col = 2)

#------------------------------------------------------------------------------
# 4.2 Models Mi where expected moisture is a polynomial function (upto power i)
# of protein and the moisture is normally distributed
```

```r
for (i in 1:6) {
  plot(tecator$Moisture, tecator$Protein ^ i)
}

#------------------------------------------------------------------------------
# 3. Fit Models Mi, i = 1,2,3,4,5,6 for training and test datasets

# No of observations in the dataset
n <- dim(tecator)[1]

# Divide dataset into training and test data
set.seed(12345)
index <- sample(1:n, floor(n / 2))
train <- tecator[index, ]
test  <- tecator[-index, ]

MSE_training   <- numeric(6)
MSE_validation <- numeric(6)

for(i in 1:6) {
  # For Model Mi: Mosture ~ Protein ^ i, calculate the training and
  # validation MSE on training and test datasets

  # Training dataset
  linreg            <- lm(formula = Moisture ~ poly(Protein, i, raw = TRUE),
                          data    = train)
  Ypred             <- predict(object  = linreg,
                               newdata = train,
                               type    = "response")
  Yp                <- train$Moisture
  MSE_training[i]   <- mean((Ypred - Yp) ^ 2)

  # Test dataset
  Ypred             <- predict(object  = linreg,
                               newdata = test,
                               type    = "response")
  Yp                <- test$Moisture
  MSE_validation[i] <- mean((Ypred - Yp) ^ 2)
}

# Plot training and validation MSE vs highest degree of protein
# polynomial function
ggplot2::ggplot(mapping = ggplot2::aes(x = 1:6, y = MSE_training)) +
ggplot2::geom_point(shape = 1) +
ggplot2::theme_classic()       +
ggplot2::theme(plot.title   = ggplot2::element_text(hjust = 0.5),
               panel.border = ggplot2::element_rect(fill = NA))  +
ggplot2::geom_smooth(method = stats::loess, se = FALSE, col = 2) +
ggplot2::labs(x = "Highest degree of Protein function",
              y = "Training MSE")

ggplot2::ggplot(mapping = ggplot2::aes(x = 1:6, y = MSE_validation)) +
ggplot2::geom_point(shape = 1) +
ggplot2::theme_classic()       +
ggplot2::theme(plot.title   = ggplot2::element_text(hjust = 0.5),
               panel.border = ggplot2::element_rect(fill = NA))  +
ggplot2::geom_smooth(method = stats::loess, se = FALSE, col = 4) +
ggplot2::labs(x = "Highest degree of Protein function",
              y = "Validation MSE")

#-----------------------------------------------------------------------------
# 4.4 Variable selection using stepAIC
linreg  <- lm(formula = Fat ~ . - Sample - Protein - Moisture,
              data    = tecator)
stepAIC <- MASS::stepAIC(object    = linreg,
                         direction = "both",
                         trace     = FALSE)

#-----------------------------------------------------------------------------
# 4.5 Ridge Regression
ridgereg <- glmnet::glmnet(x      = as.matrix(tecator[ ,2:101]),
                           y      = tecator$Fat,
                           family = "gaussian",
                           alpha  = 0)
plot(ridgereg, xvar="lambda", label=TRUE)

#-----------------------------------------------------------------------------
# 4.6 Lasso Regresssion
lassoreg <- glmnet::glmnet(x      = as.matrix(tecator[ ,2:101]),
                           y      = tecator$Fat,
                           family = "gaussian",
                           alpha  = 1)
plot(lassoreg, xvar="lambda", label=TRUE)

#-----------------------------------------------------------------------------
# 4.7 Cross validation to find optimal LASSO model
lasso.cv <- glmnet::cv.glmnet(x            = as.matrix(tecator[ ,2:101]),
                              y            = tecator$Fat,
                              family       = "gaussian",
                              alpha        = 1,
                              lambda       = seq(0, 7, 0.001),
                              type.measure = "mse")

# Minimum and Optimum lambda
lambda.min <- which(lasso.cv$lambda == lasso.cv$lambda.min)
lambda.1se <- which(lasso.cv$lambda == lasso.cv$lambda.1se)

# Number of features selected for optimum lambda
feat_optimum <- lasso.cv$nzero[lasso.cv$lambda == lasso.cv$lambda.1se]
cv_optimum   <- lasso.cv$cvm[lasso.cv$lambda == lasso.cv$lambda.1se]

# CV score w.r.t lambda
plot(x    = log(lasso.cv$lambda),
     y    = lasso.cv$cvm,
     main = "CV score w.r.t Lambda",
     xlab = "Log Lambda",
     ylab = "Mean-Squared Error",
     col  = "red" )
```